



ARCHITECTUURVOORSTEL — TER REVIEW

MarketingOost frontend

Eén codebase die elke MarketingOost-site bouwt. Dit document beschrijft hoe het project is opgezet, hoe we pagina's renderen en hoe meertaligheid werkt. Pagina's, content en de Laravel-kant vallen erbuiten.

Framework	Next.js 16, App Router
Rendering	Statisch per taal, bijwerken bij backend-wijzigingen
Data	REST naar de Laravel API, alleen server-side
Meertaligheid	next-intl, standaardtaal zonder taalprefix
Multi-site	Eén build per site, merk vastgezet bij de build
Hosting	Nog te kiezen — zie de twee opties
Opgesteld door	Blue Flamingos

Opzet

Vier lagen, elk met een eigen verantwoordelijkheid.

De frontend is opgedeeld in vier lagen. Elke laag weet alleen wat hij nodig heeft: een component kent geen merk en geen taal, maar vraagt om een token en om een vertaalde tekst. Welke waarde daaruit komt, bepaalt de laag eronder. Hierdoor kan er makkelijker een tweede website worden opgezet zonder aan de componenten te zitten.



LAAG	BEPAALT	VERANDERT BIJ
Merklaag	Hoe een site eruitziet	Een nieuwe of gewijzigde merkdefinitie
Componenten	Hoe iets werkt: toetsenbord, schermlezer, gedrag	Een wijziging voor alle sites tegelijk
Blocks	Welke onderdelen een redacteur kan plaatsen	Een nieuw blocktype in het CMS
Routes	Hoe een adres bij een pagina komt: één route vangt elke slug uit het CMS op	Vrijwel nooit — een nieuwe pagina vraagt geen nieuwe route

AUTOMATISCH GECONTROLEERD

Een kleur of lettergrootte die rechtstreeks in componentcode staat, laat de build falen. Zonder die controle sluipt de vormgeving van de ene site in een andere site.

MOBILE FIRST

Ruim 70% van de bezoekers komt via mobiel. De mobiele weergave is daarom de basis en het eerste beoordelingsmoment; grotere schermen zijn de uitbreiding.

Data en het contract

De API bepaalt de vorm. Wij zorgen dat een wijziging de site niet omver haalt.

Alle content komt via REST uit de Laravel API. Die roepen we aan vanaf de server: elke call heeft twee sleutels nodig, en die horen niet in de browser.

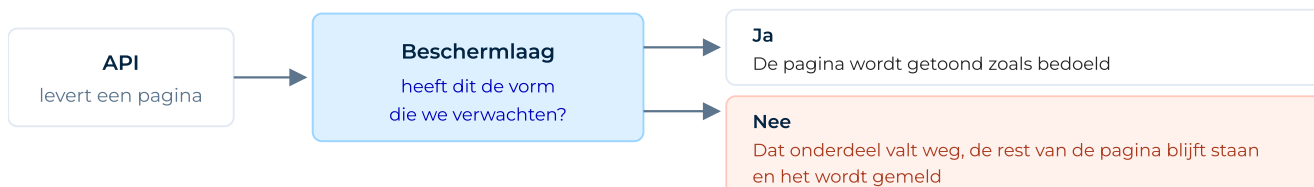
Waar de typen vandaan komen

De API publiceert een beschrijving van zichzelf, inmiddels tot op de inhoud van elk blocktype. Daaruit genereren we de code die de API aanroept, de typen en de controles, zodat het contract één bron heeft en wij niets overtypen. Eerder legden we de inhoud van een block met de hand vast, omdat de beschrijving die openliet. Die handmatige versie was al uit de pas gelopen zonder dat iemand het zag; dat kan nu niet meer.

Wat we niet doen, is het contract in de frontend bepalen. Het CMS bepaalt welke blocks en welke varianten er bestaan, en elk block heeft er minstens één (`default` als er maar één look is). Elke variant krijgt bij ons een eigen component, zodat een nieuw ontwerp voor de ene variant de andere niet raakt. Een look die het CMS niet kent, bouwen we niet zelf na: een redacteur zou hem toch niet kunnen kiezen.

De bescherm laag

Bij binnenkomst controleren we of een antwoord de vorm heeft die we verwachten. Klopt er iets niet, dan valt dat ene onderdeel weg in plaats van dat de pagina stukgaat. Een hernoemd veld zien we al zodra we de nieuwe beschrijving binnenhalen, op één plek; geen component kent de namen van het CMS.



WAAROM ZO

Een site die één blok mist is beter dan een site die niets toont. De keuze is bewust: liever zichtbaar incompleet dan onzichtbaar kapot. Wat wegviel wordt gemeld, zodat het opgelost kan worden in plaats van dat het maanden onopgemerkt blijft.

Renderen

Alle pagina's worden vooraf gebouwd, per taal.

Wat een bezoeker te zien krijgt, hangt niet van die bezoeker af. Het merk staat vast bij de build en de taal staat in de URL. Content verandert wanneer een redacteur publiceert, niet per seconde. Daarom bouwen we elke pagina vooraf en serveren we kant-en-klare HTML.

De twee alternatieven vielen af. Bij het bouwen van elke pagina per bezoek staat de bezoeker te wachten op de API, en gaat de site plat als de API dat doet. Bij vernieuwen op een vaste interval is elke keuze gokken: te kort is verspilling, te lang betekent verouderde content. Publiceren is een preciezer signaal.

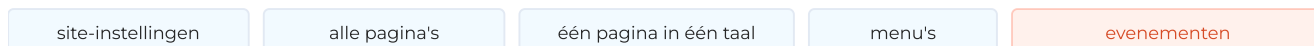


Bijwerken bij een wijziging

Als een redacteur publiceert, stuurt het CMS een bericht naar de frontend. Daarin staat wat er veranderde. Alleen die onderdelen worden opnieuw opgehaald; de rest blijft staan. Er komt geen nieuwe deploy aan te pas en de bezoeker merkt er niets van behalve dat de wijziging er is.



Wat apart bijgewerkt kan worden



EVENEMENTEN WIJKEN AF

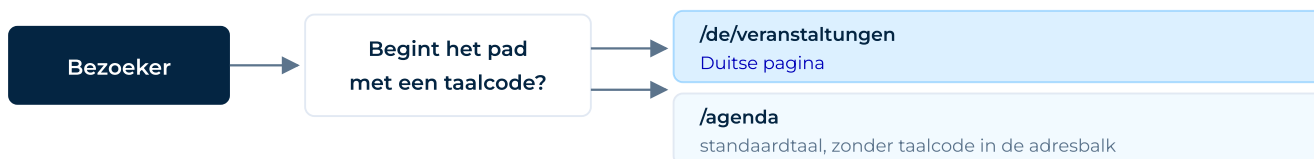
Die komen uit een externe bron en veranderen zonder dat er iemand publiceert. Bovendien is een lijst als "vandaag" morgen niet meer waar, ook als er niets aan de data verandert. Daarom werken we die apart bij: na een synchronisatie met die bron, en één keer per nacht als de datum verschuift.

Meertaligheid

De URL bepaalt de taal.

De backend levert content al vertaald aan: een gepubliceerde pagina is één taal. De frontend regelt twee dingen. Welke URL welke taal serveert, en de teksten die niet uit het CMS komen.

Hoe de URL's eruitzien



We kijken niet naar de browsertaal

Een Duitse browser die naar / gaat, krijgt de Nederlandse site. Zou de browsertaal meebeslissen, dan is de Nederlandse site niet meer op zijn eigen adres te bereiken en werkt de taalwisselaar niet meer terug.

Uit het CMS

Alle content, al vertaald: koppen, teksten, menu's en de SEO-velden. Ook het webadres per taal, want dat verschilt. Een taalwisselaar mag daarom nooit zelf het huidige adres omschrijven; de API levert per taal het juiste adres mee.

Uit de frontend

Teksten die bij de interface horen: knoplabels, foutmeldingen bij een formulier, sorteer- en filteropties, en de teksten die alleen een schermlezer voorleest. Die staan per taal in een vertaalbestand. Alles wat een redacteur zou willen aanpassen, hoort niet hier maar in het CMS.

Wat een extra taal kost

ONDERDEEL	PER TAAL
Pagina's	Eén extra set vooraf gebouwde pagina's. De build duurt langer, de site wordt niet trager
Content	Niets extra's: de backend levert de vertaalde versie
Interfaceteksten	Eén vertaalbestand
Merk	Niets. Welke talen een site heeft, staat in de merkdefinitie

TOEGANKELIJKHEID

De taalwisselaar noemt elke taal in die taal zelf — "Deutsch", niet "DE" — en geeft per optie aan om welke taal het gaat, zodat een schermlezer van stem wisselt. Zoekparameters in de URL blijven behouden, zodat iemand die via een campagnelink binnenkomt die niet verliest door van taal te wisselen.

Structuur en kwaliteit

Waar de code staat, en hoe we controleren dat hij werkt.

Indeling

src/	
app/	routes en layouts
blocks/	per blocktype
components/	gedeelde UI
brands/	merkdefinities
styles/	stylesheets
lib/	data en hulpcode
i18n/	routing, vertalingen
openapi/	beschrijving van de API
scripts/	generatoren
tests/	component- en logicatests
e2e/	tests in een echte browser
docs/	.adrs/ besluiten

De bestanden die uit de merkdefinities gegenereerd worden, passen we niet met de hand aan. De build faalt als ze uit de pas lopen.

Eén build per site

Het merk — en dus de styling — wordt vastgezet met een env-variabele (`BRAND`) en staat dus vast op het moment van bouwen.

Nieuwe site toevoegen

- 1 Merkdefinitie aanmaken
- 2 Bestanden laten genereren
- 3 Bouwen met dat merk

Geen nieuw project, geen kopie van de componenten.

Wat we controleren

Dit draait bij elke wijziging, voordat er iets samengevoegd wordt.

CONTROLE	WAT HET TEGENHOUDT
Types	Een veld dat niet bestaat, of een waarde van de verkeerde soort
Codecontrole	Vormgeving die niet uit de merklaag komt, en links die de taal van de bezoeker kwijtraken
Componenttests	Gedrag per component en per blocktype: wat er getoond wordt, en wat er gebeurt bij een klik. Wat alleen op een echte server bestaat, wordt in een browser getest
Merkdefinities	Gegenereerde bestanden die niet meer overeenkomen met hun bron
API-contract	Gegenereerde code die niet meer bij de beschrijving van de API past, en een blocktype dat het CMS niet kent
Build per site	Een pagina die ongemerkt niet meer vooraf gebouwd wordt. De site blijft gewoon werken, maar die pagina wordt dan bij elk bezoek opnieuw opgebouwd, met telkens een aanroep naar de API

Hosting

Nog niet gekozen. Twee richtingen.

De applicatie verandert niet door deze keuze: dezelfde build draait op beide. Wat verschilt is wie het beheer doet en wat het kost.

Beheerd platform

Bijvoorbeeld Vercel, de partij achter Next.js.

Voordelen

Uitrollen gebeurt vanzelf bij een wijziging, met een preview-omgeving per wijziging. Meerdere instanties naast elkaar werkt zonder extra inrichting, dus het schaaft mee zonder ons werk.

Nadelen

Kosten lopen op per site en per bezoek, en zijn lastiger vooraf vast te zetten. We zijn afhankelijk van één leverancier, en het draait buiten eigen beheer.

Zelf hosten

Op eigen servers, eventueel met een beheerlaag als Coolify.

Voordelen

Voorspelbare kosten, en alles blijft in eigen beheer. Meerdere sites kunnen op dezelfde server staan. Geen afhankelijkheid van één leverancier.

Nadelen

Uitrollen, certificaten, monitoring en back-ups regelen we zelf. Meerdere instanties van dezelfde site vraagt extra inrichting; zonder dat is één instantie per site het uitgangspunt.

Afbeeldingen

Een bezoeker moet geen foto van vier megapixel binnenkrijgen voor een plaatje van 400 pixels breed. Het framework verkleint afbeeldingen zelf bij de eerste opvraag en levert ze in moderne bestandsformaten, zonder extra configuratie. De verkleinde versies komen in dezelfde cache-opslag terecht die de site toch al gebruikt; de enige voorwaarde aan de hosting is dat die opslag een herstart overleeft.

Hoe dan ook te regelen: domeinnamen per site voor productie, acceptatie en preview, en het adres waar het CMS zijn bericht naartoe stuurt per omgeving.

Waar we akkoord op vragen

Zes besluiten. Daarna kunnen de vervolgtickets opgepakt worden.

- 1 Next.js met de App Router als basis.** De aanroepen naar de API blijven op de server, en de taal zit in het routepad in plaats van in een uitzondering per pagina.
- 2 Pagina's vooraf bouwen, bijwerken zodra de backend wijzigt.** Geen wachttijd op de API per bezoek, en geen verouderde content door een vaste interval.
- 3 REST naar de Laravel API, alleen vanaf de server.** De sleutels bereiken de browser niet. De API bepaalt de vorm; onze typen en controles worden uit zijn beschrijving gegenereerd.
- 4 Blocks en varianten komen uit het CMS.** Elke variant is een eigen component; een look die het CMS niet kent, is eerst een vraag aan de backend.
- 5 Eén build per site.** Een nieuwe site is een merkdefinitie, geen nieuw project en geen kopie van de componenten.
- 6 De indeling en de controles uit dit document.** Inclusief de controle die vormgeving buiten de componenten houdt.

Wat nog open staat

Hosting	Beheerd platform of zelf hosten — en daarmee hoe we het pakketteren
Analytics en cookies	De site-instellingen leveren de ids al aan; de koppeling is een apart onderwerp